**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Лабораторная работа №4**

**«Численные методы»**

**Вариант 8**

Ёды Никиты Дмитриевича

студента 3 курса, 6 группы

специальность «прикладная математика»

Преподаватель:

Будник А.М.

**Постановка задачи**

Найти приближенное решение задачи Коши

,

на сетке узлов при 10-ти разбиениях отрезка интегрирования применяя следующие методы:

1. Неявный метод Эйлера. Для его реализации использовать алгоритм метода Ньютона.
2. Метод Рунге-Кутта при
3. Экстраполяционный метод Адамса 3-го порядка с началом таблицы, построенным по соответствующему методу последовательного повышения порядка точности.

Для проведения анализа полученных результатов необходимо:

а. Используя таблицу приближенных результатов, получить погрешности методов 1 и 2, считая “точным решением” метод 3.

б. На основе полученных численных результатов сделать вывод о точности каждого метода.

**Метод Эйлера**

Метод Эйлера выглядит следующим образом:

Рассмотрим главный член локальной погрешности:

Таким образом порядок локальной погрешности – 2. Погрешность метода 𝑂(ℎ).

**Реализация**

|  |
| --- |
| import numpy as np  def f(x, u):      return (u + np.sqrt(x\*\*2 + u\*\*2)) / x  u0 = 0  n = 10  a = 1  b = 1.5  def euler(a, b, n, f, y0):      h = (b - a) / n      x = [a + h\*i for i in range(n+1)]      y = [y0]      for i in range(n):          y.append(y[i] + h\*f(x[i], y[i]))      return x, y  sol = euler(a, b, n, f, u0)  print('Метод Эйлера')  for x, y in zip(sol[0], sol[1]): print(f'u({x:.2f}) = {y:.8f}') |

**Вывод по полученным результатам**

|  |
| --- |
| Метод Эйлера  u(1.00) = 0.00000000  u(1.05) = 0.05000000  u(1.10) = 0.10243761  u(1.15) = 0.15731020  u(1.20) = 0.21461541  u(1.25) = 0.27435107  u(1.30) = 0.33651525  u(1.35) = 0.40110616  u(1.40) = 0.46812222  u(1.45) = 0.53756196  u(1.50) = 0.60942406 |

**Метод последовательного повышения порядка точности 2-го порядка**

Для реализации метода будут использоваться формулы:

Так как метод второго порядка, то локальная погрешность – O(ℎ3), а погрешность метода O(ℎ2).

**Реализация**

|  |
| --- |
| import numpy as np  def f(x, u):      return (u + np.sqrt(x\*\*2 + u\*\*2)) / x  u0 = 0  n = 10  a = 1  b = 1.5  def increase\_accuracy(a, b, n, f, y0):      h = (b - a) / n      x = [a + h \* i for i in range(n + 1)]      y = [y0]      for j in range(n):          y\_half = y[j] + h / 2 \* f(x[j], y[j])          y.append(y[j] + h \* f(x[j] + 1 / 2 \* h, y\_half))      return x, y  sol = increase\_accuracy(a, b, n, f, u0)  print('Метод последовательного повышения порядка точности')  for x, y in zip(sol[0], sol[1]):      print(f'u({x:.2f}) = {y:.8f}') |

**Вывод по полученным результатам**

|  |
| --- |
| Метод последовательного повышения порядка точности  u(1.00) = 0.00000000  u(1.05) = 0.05123438  u(1.10) = 0.10496800  u(1.15) = 0.16120086  u(1.20) = 0.21993295  u(1.25) = 0.28116428  u(1.30) = 0.34489484  u(1.35) = 0.41112464  u(1.40) = 0.47985367  u(1.45) = 0.55108194  u(1.50) = 0.62480944 |

**Интерполяционный метод Адамса**

Интерполяционный метод Адамса третьего порядка использует интерполяционный полином второй степени для аппроксимации функции на предыдущих шагах. Полином имеет вид:

Используя этот полином, мы можем аппроксимировать значение функции на следующем шаге :

|  |
| --- |
| import numpy as np  # Метод Рунге-Кутты для построения начальных значений  def runge\_kutta(f, x0, y0, h, n):      x\_values = [x0]      y\_values = [y0]      for i in range(n):          k1 = f(x\_values[-1], y\_values[-1])          k2 = f(x\_values[-1] + h / 2, y\_values[-1] + h / 2 \* k1)          k3 = f(x\_values[-1] + h / 2, y\_values[-1] + h / 2 \* k2)          k4 = f(x\_values[-1] + h, y\_values[-1] + h \* k3)          x\_values.append(x\_values[-1] + h)          y\_values.append(y\_values[-1] + h / 6 \* (k1 + 2 \* k2 + 2 \* k3 + k4))      return x\_values, y\_values  # Функция правой части дифференциального уравнения для метода Рунге-Кутты  def f(x, u):      return (u + np.sqrt(x\*\*2 + u\*\*2)) / x  # Метод простой итерации для интерполяционного метода Адамса 3-го порядка  def adams\_method(x\_values, y\_values, h):      # x\_values: значения x      # y\_values: значения y, соответствующие x\_values  # h: шаг      n = len(x\_values)      # Инициализация значений для метода Адамса (по методу Рунге-Кутты)      u\_values = y\_values      # Вычисление остальных значений методом простой итерации      for i in range(n - 1, 2, -1):          # Начальное приближение для u\_{i+1}          u\_next = u\_values[i]          # Метод простой итерации          max\_iterations = 100          epsilon = 1e-7          for \_ in range(max\_iterations):              u\_next\_old = u\_next              u\_next = y\_values[i] + h / 12 \* (                          5 \* f(x\_values[i], u\_next) + 8 \* f(x\_values[i - 1], u\_values[i - 1]) - f(x\_values[i - 2],                                                                                                   u\_values[i - 2]))              if abs(u\_next - u\_next\_old) < epsilon:                  break          u\_values.append(u\_next)      return u\_values  # Начальные условия  x0 = 1  y0 = 0  h = 0.05  n = 10  # Построение начальных значений методом Рунге-Кутты  x\_values, y\_values = runge\_kutta(f, x0, y0, h, n)  # Вычисление остальных значений методом Адамса  u\_values = adams\_method(x\_values, y\_values, h)  print("Метод Адамса")  for i in range(len(x\_values)):      print(f" u({x\_values[i]:.2f}) = {u\_values[i]:.8f}") |

**Вывод по полученным результатам**

|  |
| --- |
| Метод Адамса  u(1.00) = 0.00000000  u(1.05) = 0.05125000  u(1.10) = 0.10500000  u(1.15) = 0.16124999  u(1.20) = 0.21999999  u(1.25) = 0.28124999  u(1.30) = 0.34499998  u(1.35) = 0.41124998  u(1.40) = 0.47999998  u(1.45) = 0.55124997  u(1.50) = 0.62499997 |

**Анализ методов**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Метод Эйлера | Метод последовательного повышения порядка точности | Метод Адамса |
| 1.00 | 0.00000000 | 0.00000000 | 0.00000000 |
| 1.05 | 0.05000000 | 0.05123438 | 0.05125000 |
| 1.10 | 0.10243761 | 0.10496800 | 0.10500000 |
| 1.15 | 0.15731020 | 0.16120086 | 0.16124999 |
| 1.20 | 0.21461541 | 0.21993295 | 0.21999999 |
| 1.25 | 0.27435107 | 0.28116428 | 0.28124999 |
| 1.30 | 0.33651525 | 0.34489484 | 0.34499998 |
| 1.35 | 0.40110616 | 0.41112464 | 0.41124998 |
| 1.40 | 0.46812222 | 0.47985367 | 0.47999998 |
| 1.45 | 0.53756196 | 0.55108194 | 0.55124997 |
| 1.50 | 0.60942406 | 0.62480944 | 0.62499997 |

Таблица погрешности метода Эйлера и метода последовательного повышения порядка точности:

|  |  |  |
| --- | --- | --- |
|  | Метод Эйлера | Метод последовательного повышения порядка точности |
| 1.00 | 0.00000000 | 0.00000000 |
| 1.05 | 0,00125000 | 0,00001562 |
| 1.10 | 0,00256239 | 0,00003200 |
| 1.15 | 0,00393979 | 0,00004913 |
| 1.20 | 0,00538458 | 0,00006704 |
| 1.25 | 0,00689892 | 0,00008571 |
| 1.30 | 0,00848473 | 0,00010514 |
| 1.35 | 0,01014382 | 0,00012534 |
| 1.40 | 0,01187776 | 0,00014631 |
| 1.45 | 0,01368801 | 0,00016803 |
| 1.50 | 0,01557591 | 0,00019053 |

|  |  |
| --- | --- |
| Метод Эйлера | Это один из самых простых численных методов для решения ОДУ.  Он использует линейную аппроксимацию производной и явный шаг по времени.  Метод Эйлера обладает первым порядком точности, что означает, что ошибка метода уменьшается пропорционально квадрату шага. |
| Метод последовательного повышения порядка точности: | Этот метод является общим методом, который позволяет увеличить порядок точности метода, используя более высокие степени интерполяции или аппроксимации.  Он позволяет увеличить точность при сохранении простоты реализации и является более точным, чем метод Эйлера.  Однако более высокие порядки требуют большего количества вычислений и могут стать более неустойчивыми, особенно при использовании явных методов. |
| Методы Адамса: | Эти методы являются явными методами, использующими интерполяционные полиномы для аппроксимации функции на предыдущих шагах.  Методы Адамса обладают более высокими порядками точности по сравнению с методом Эйлера и могут быть более стабильными при правильном выборе шага интегрирования.  Однако они могут стать неустойчивыми при больших шагах интегрирования или при решении жестких систем ОДУ. |